Web Services Workflow for Online Data Visualization and Analyis in Giovanni

S. Berrick, M. Butler, J. Farley, J. Hosler, L. Lighty, H. Rui
NASA Goddard Space Flight Center
Code 610.2, Greenbelt, MD, 20771

***Abstract*— The Goddard Earth Sciences (GES) Data and Information Services Center (DISC) is undertaking an architectural upgrade of its GES-DISC Interactive Online Visualization ANd aNalysis Infrastructure or "Giovanni" to take it from a synchronous CGI application to an asynchronous Web services workflow management system. Because Giovanni has been enormously successful from a user standpoint, increased interest in its usage and user feedback resulted in new requirements for extensibility and scalability. Giovanni's success has also spawned numerous new science requirements. The assessment of the new requirements has determined that they cannot be implemented well without architectural revisions to Giovanni. This has resulted in a new architectural concept denoted as Giovanni 3. The goals of the Giovanni 3 architecture include support for remote data via standard protocols such as OPeNDAP, GrADS Data Server (GDS), and Web services; processing and rendering via standard Web services; workflow management that allows users to modify and save their own workflows; a customizable and context-sensitive component based interface; and asynchronous processing.**

Keywords-distributed data analysis; data visualization; Web applications; online

I. INTRODUCTION

A continuing challenge at NASA is how to provide science users with a meaningful way to distill knowledge from the petabytes of multi-sensor Earth science data available without requiring users to first download large volumes of data, understand their formats and structure, and only then perform the necessary analysis. Traditional approaches such as data mining are key, but services that allow users to explore data through analysis and visualization are equally important. To address this need, the Goddard Earth Sciences Data and Information Services Center (GES DISC) developed the GES-DISC Interactive Online Visualization ANd aNalysis Infrastructure or "Giovanni". The first instance of Giovanni debuted in the summer of 2002 with the TRMM (Tropical Rainfall Measuring Mission) Online Visualization and Analysis System (TOVAS) targeted to the precipitation science community. TOVAS was extremely successful and led the way for additional Giovanni instances targeting other Earth science communities.

The initial architecture of Giovanni (now, referred to as Giovanni 2) was based on CGI Perl scripts, HTML templates, and GrADS (Grid Analysis and Display System) [1] scripts for doing the actual processing and image rendering. A series of hierarchical configuration files defined the datasets and parameters, functions, and image rendering options for each Giovanni instance. In principle, the addition of a new data set or function to an existing Giovanni interface was as simple as making a few modifications to the configuration files followed by a regeneration of the instance using a Perl script. In practice, however, the process was not so simple.

II. GIOVANNI 2 LIMITATIONS

The Giovanni 2 architecture was deployed widely at the GES DISC [2] to ultimately support nine instances and was enormously successful in terms of user satisfaction. With requirements expansion because of the increased science community interest, the architectural limitations with respect to extensibility and scalability became apparent. Incorporating new requirements within the current Giovanni architecture appeared to be increasingly expensive in maintenance costs. Moreover, the Giovanni 2 architecture was needlessly restrictive in dealing with science requirements:

A. Configuration Complexity

The hierarchical configuration files supporting the Giovanni instances grew ever more complex as new features, based on new science requirements, were added to the interface to provide users with an increasing set of customization options. In addition, the user had no control over the options or what data were available; these all had to be preset by the developers.

B. Supports Only GrADS

The scripting language used in Giovanni, GrADS, supported both the processing and the image rendering. Although well suited for Level 3 gridded data, GrADS has limitations in supporting Level 2 data, something our data users were eager to have available in Giovanni.

C. Synchronous Interface

The synchronous nature of the Web interface meant that processing and image rendering had to be fast enough to not cause an HTTP session time-out. Workarounds included keeping the data on the local machine or preprocessing the

data to improve access performance; and restricting the amount of processing available to the user (either by providing only coarse resolution data or by restricting the spatial area).

D. No Pluggable Interface

The infrastructure did not provide a means for science users or even Giovanni maintainers to plug in new processing or image rendering code according to well-defined documented interfaces. This resulted in increased costs in extending the processing capabilities of Giovanni.

III. GIOVANNI 3 REQUIREMENTS

As a result of the described Giovanni 2 limitations and other considerations, a new Giovanni architectural vision emerged known as Giovanni 3. The requirements for Giovanni 3 are:

- Clear separation between infrastructure and processing/rendering - Further, the architecture should be agnostic with respect to processing or image rendering software.
- Data location transparency – The architecture should work with local or remote data via standard protocols such as OPeNDAP (Open-source Network for a Data Access Protocol) [3].
- Asynchronous – The architecture should move beyond the restrictive synchronous CGI paradigm to allow more complex processing supported by RSS feeds to alert a user when the product is available.
- Workflow – Processing should be managed using a workflow paradigm that allows the user to modify existing workflows, create new ones, and save them for future executions.
- Service Oriented Architecture – The architecture should use well-defined XML-based interfaces for adding/deleting data sets or parameters and adding/deleting processing or image rendering functions.
- User Configurable Interface – The architecture should be based on configurable, context-sensitive components and allow the user to create, modify, and save the interface configurations. In short, support a "My Giovanni" concept.
- Data Lineage – The architecture should be able to provide the user with the detailed processing steps involved in going from the initial product to the final product or image [5].

IV. ARCHITECTURE

The Giovanni 3 architecture represented as a Functional Diagram is shown in Fig. 1.

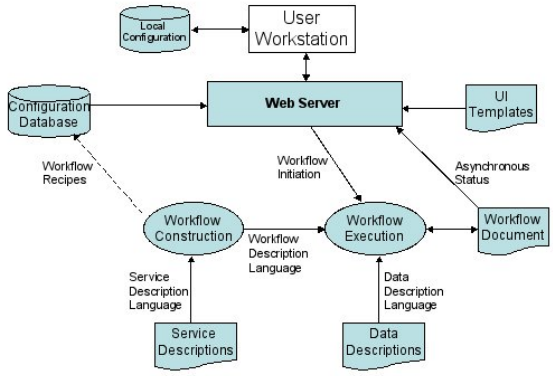

Fig. 1 – Giovanni 3 Functional Architecture

The Giovanni 3 architecture is designed to deal with specific Giovanni 2 limitations: To ease configuration complexity by providing the Giovanni support staff and users with a simple way to add new instances, services, and datasets; and to allow for more complex processing by developing an asynchronous form of processing to avoid HTTP time-outs.

The Giovanni 3 Graphical User Interface (GUI) utilizes a Web server, CGI scripts, style sheets, HTML templates, server side includes (SSI), and a configuration database to create a dynamic, interactive environment for the user. The Web page content is dynamically generated using a combination of CGI and database scripts. Rules are set using database schemas to only allow the pertinent parameters and services (processing and image rendering functions) to be used for a specific instance.

Currently scripts are used to manage the database configuration. In the future, however, there will be an XML representation of the database. Users will be able to save this XML-based database to their local machines and create their own "My Giovanni" configurations, and there will be a GUI in place to allow users to manage them. It should be noted that a GUI is not required to execute a workflow. It can also be executed from the command line or, in the future, via a Web Service.

A workflow is a series of discrete processing steps that together generate a product (*e.g.* an image) where each processing step is defined by its inputs, outputs, and a transformation in the form of an algorithm. The workflow management layer of Fig. 1 is logically divided into workflow construction and workflow execution. Workflow construction is performed as an off-line activity, well in advance of executing the workflow via a workflow engine.

Workflow construction allows the user to build a network of pre-defined services which will be executed at runtime. Each service in the workflow is defined in Service Description Language (SDL), our XML representation of the service to be executed. SDL provides the capability to execute command-line programs, Web Services via WSDL, as well as Perl and Python functions. A workflow is constructed by specifying the services to be executed in a workflow recipe with an XML representation. The SDL representation of a service provides:

- A URL to an XML Schema (XSchema format) that describes the inputs and outputs of the service.
- A binding section that specifies how to execute the service.
- A URL to a description of the processing provided by the service. This description is used by the GUI to present the data lineage of the data produced by the workflow.

Once a workflow recipe has been constructed, information required to execute the workflow is exported to the Configuration Database where it is used by the user interface (UI) software to dynamically construct an appropriate graphical user interface (GUI). When a user comes to the Giovanni 3 landing page, an initial script is run to access the database. The user is redirected to a Web page that shows all of the currently available Giovanni instances. When the user clicks on an instance, another script is launched which generates a page showing the name of the instance, an introductory paragraph about the data, and a Web form that lists all of the available parameters, services, and associated options for that instance.

When the user has selected the inputs for the workflow from the GUI, the UI software creates an XML representation of the inputs and initiates execution of the appropriate workflow.

During this waiting period, the user is redirected to an execution status page. If the user chooses not to wait for the results, the system can automatically notify the user of processing completion through the use of Really Simple Syndication (RSS). During workflow processing, Giovanni updates a unique user RSS XML feed to indicate changes in the workflow state. Upon completion of the workflow processing, a link to the resultant product (typically an image) is placed into the RSS feed. The use of RSS provides the user with an asynchronous method of obtaining their product, and eliminates dependencies on HTTP time limitations. In cases where the processing is fast, the RSS mechanism is bypassed and the user gets the resultant product within the current HTTP session; the processing appears to be synchronous.

Currently a workflow engine does not perform workflow execution. Our intent is not to develop our own workflow engine, but to reuse an existing engine. We performed a survey of currently available engines, with little luck in finding a perfect match. At present we are targeting the SciFlo engine that is being developed at JPL by the Global Environmental and Earth Science Investigation System (GENESIS) Project [4]. At the time of this writing, the SciFlo engine had not yet been released. In the meantime, we have constructed a workflow script generator (genscript) that can read a workflow recipe file (styled according to that defined in SciFlo [4]) and generate a Perl script that can execute the workflow. This allows us to continue development with a focus on the services to be provided and how well they work within a workflow framework, without actually having a workflow engine.

When the GUI executes a workflow script, an XML workflow document is created for that specific execution. The workflow document contains all information needed to execute the workflow including user inputs as well as a description of the network of services to be executed. As each service is executed, it is responsible for updating its status and outputs in the workflow document. This allows the workflow document to serve as the state repository for the executing workflow. A variety of XSL transforms are provided to present information about the executing workflow. From the GUI, the user can request the current status of the workflow. At that point, an XSL style-sheet is executed on the workflow document which produces an HTML rendering of the current status. Similarly, an XSL style-sheet is provided which produces an RSS formatted XML document. Users can set up their favorite RSS reader to monitor the status of their request. Each time the RSS reader attempts to load the RSS document, the style-sheet is re-executed on the workflow document to produce the latest status in RSS format.

Simple services can be composited into more complex services. For example, a variety of “averaging” services are available in the Giovanni 3 system (modeled after averaging algorithms in Giovanni 2). The different instances of an averaging service can operate on specific data structures. The implementation of an averaging service that operates on Level 3 regularly spaced gridded data is very different from one that operates on Level 2 swath data. In the Giovanni system, both of these services conform to the same schema that specifies the inputs and outputs of the averaging operation without any details of the implementation. This allows us to build a composite averaging service, which can call the appropriate low-level averaging service based upon the structure type of the data to be averaged.

In order to automate the composite service, a detailed description of the data is provided in a Data Description Language (DDL). The DDL for a data product describes the contents, structure, and relationships of a specified data product. In the case of the composite averaging service, by looking in the DDL for the structure of the input data, the composite service can automatically call the correct low-level service.

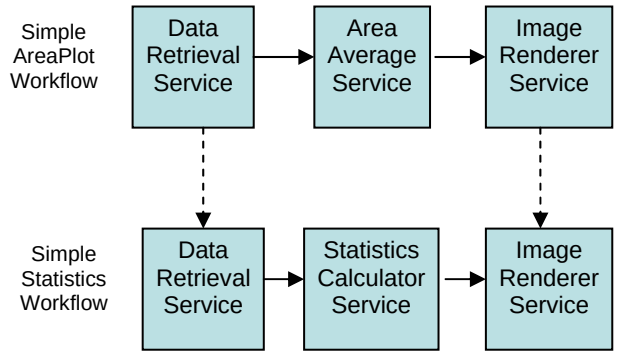

Fig. 2 – Sample Workflow Schematics

Fig. 2 shows a diagrammatic example of a simple workflow, and the simplicity of constructing workflows from pre-defined services. The first step in the workflow is to call the data retrieval service. This service will retrieve the data from its original location into a local cache via a call to the appropriate low-level retrieval service. The second step in the workflow calls a calculator service to compute a pixel-by-pixel average over the range of the retrieved data. The final step in the workflow is an Image Renderer that produces a plot of the area-averaged data that is suitable for display in a Web browser. As shown in the figure, the area average service can easily be replaced with a statistics calculator service that produces minimum, maximum, average, and standard deviation statistics from the input data. This new workflow would then produce image renderings of the statistics data. The data retrieval and rendering services are identical in the two workflows, but the resulting images are very different.

A variety of pre-built services are provided for construction of Giovanni workflows. These services can be categorized as:

- Data Retrieval – A composite data retrieval service is provided to fetch data from its source location into a temporary cache. Low-level services are available for retrieving data via HTTP, FTP, and OPeNDAP protocols.
- Data Transform – A variety of transform services are provided to manipulate the retrieved data. Available transforms include scaling, subsetting, and re-gridding.
- Calculation – Calculator services are provided to average, subtract, add, and generate statistics for the input data.
- Filter – Various filter services are provided to select a subset of the input data.
- Rendering – Services that render images in GIF, PNG, GeoTIFF and other formats are provided to output data in ways that are easy to display and print.
- Packaging - For users who want the raw data, instead of images, packaging services are provided to deliver the outputs of a workflow in ASCII, binary, or HDF formats.

More services will be provided over time. The list of new services to be developed is periodically re-prioritized to meet the upcoming needs of the GES DISC.

ACKNOWLEDGMENT

The GES DISC would like to acknowledge the Giovanni core team who developed Giovanni 2 without which Giovanni 3 would not exist. This team, lead by Ms. Hualan Rui consists of Timothy Dorman, Dr. Zhong Liu, Dr. Suhung Shen, Xiaoping Zhang, and Ms. Tong Zhu. Equally important are the GES DISC Atmospheric scientists Dr. James Acker, Dr. Suraiya Ahmad, Arun Gopalan, James Johnson, Dr. Gregory Leptoukh, Jason Li, Dr. Jianping Mao, Dr. Andrey Savtchenko, and Dr. William Teng,

The GES DISC wishes also to thank Dr. Yoram Kaufman (NASA GSFC) for his financial and scientific support of the MODIS Online Visualization and Analysis System (MOVAS) development effort. The Ocean Color Online Visualization and Analysis system (OOVAS) was supported by NASA HQ and Dr. Watson Gregg (NASA GSFC) through REASoN CAN 02-OES-01. The Agricultural Online Visualization and Analysis System (AOVAS) was also supported by REASoN CAN 02-OES-01.

REFERENCES

[1] GrADS Home Page, http://www.iges.org/grads/.

[2] Berrick, S., L. Pham, G. Leptoukh, Z. Liu, H. Rui, S. Shen, W. Teng, T. Zhu, 2004. Multi-Sensor Distributive On-line Processing Visualization and Analysis System Using Giovanni, IGARSS conference, Anchorage, AK Sep 15-24.

[3] OPeNDAP Home Page, http://www.opendap.org.

[4] GENESIS SciFlo Home Page, http://sciflo.jpl.nasa.gov.

[5] Bose, R., and Frew, J., 2004. Composing Lineage Metadata with XML for Custom Satellite-Derived Data Products, Sixteenth International Conference on Scientific and Statistical Database Management, Santorini Island, Greece, 21-23 June 2004.